tetrate

White Paper

The Value of Istio and Envoy

What You Get from an Istio Service Mesh.

by Zack Butcher

www.tetrate.io

Executive Summary

The transition to service-based models driving industry advancement has inadvertently led to increased complexity in service management, networking, and security... problems only a service mesh can solve. Istio is currently the industry-leading open source service mesh, and relies on an Envoy proxy to support it.

We will address the what, why, and how of an Istio—and Envoy-based service mesh. This paper will introduce you to:

- *Why* a service mesh is encouraged in service-based models
- The *basics* of its architecture
- An overview of *features* an Istio service mesh can provide
- What the service mesh is *not* going to fix

By the end, you should have a firm understanding of what an Istio service mesh is and how it works.

Intro

Do you want your teams to be able to work independently of each other for speed?

Sure!

Do you want to use several languages to serve specific needs, rather than shoe-horning a language into fixing your problem, because you can't use another?

Of course I do!

Do you want to be able to scale efficiently and faster?

Who the heck wouldn't?

How about reducing downtime and increasing resilience?

These seem like very leading questions now.

If these are questions that resonate with your needs, then you've already started down the path of moving to microservices. You've accepted that you want to have a development strategy built around organizational and business capabilities. You've decided that you want to be flexible and more scalable in areas that you decide, more productive, faster, and efficient. But...

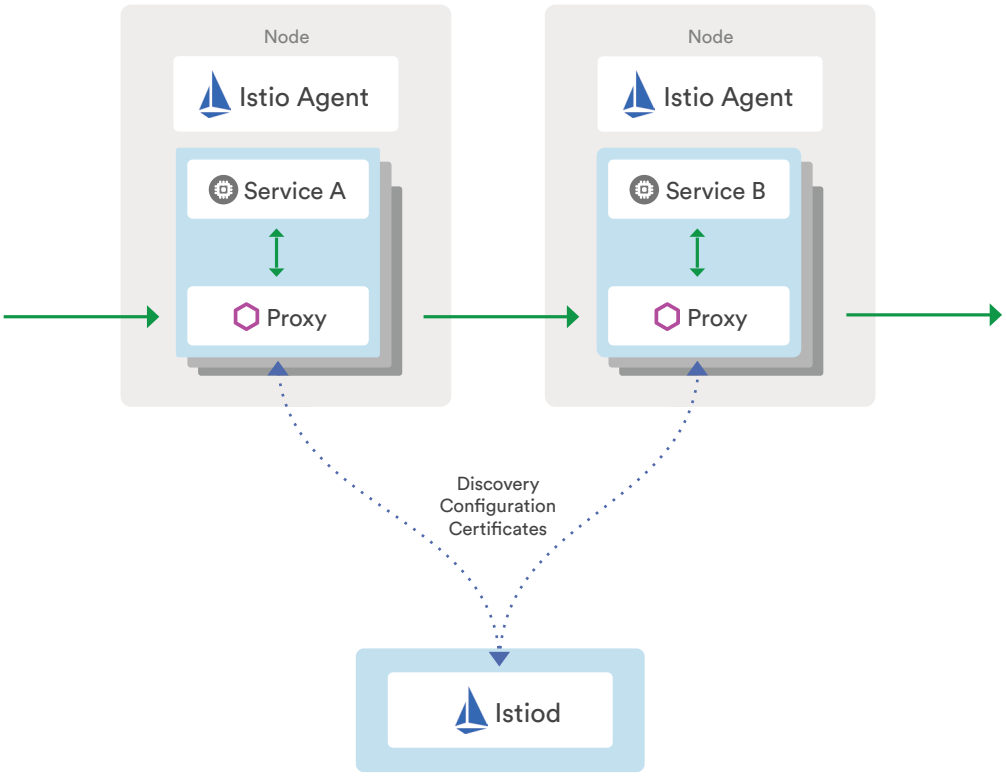
"All magic comes with a price" - *Rumplestiltskin*

Microservices have become increasingly popular because companies want to become more agile, or quick to deploy software and apps to users. But microservice architectures were designed to solve organizational problems, not engineering ones. Whether you're starting from scratch or migrating an older system to this new, exciting method of architecting an application, it comes with additional complexity and engineering concerns that need to be addressed properly to avoid abject chaos. Why is this so complex? Two reasons: environments have changed, and they're more dynamic. You can apply changes faster, but traditional networking and security approaches won't be as effective. The network is a more integral part of the application and must be managed differently.

A service mesh manages the complexity. The service mesh architecture is a direct response to the technical challenges caused by the organizational decision to move to a service-based architecture. We believe that Istio, built on Envoy, is the most comprehensive service mesh offering and the best on the market.

What is a service mesh?

A service mesh is an infrastructure layer and platform that sits on top of your service-based application to make service-service communications faster, safer and more reliable. The mesh also gives organizations a centralized, uniform way to manage, secure and connect microservices. We usually break down a service mesh into three key areas of benefit: security, traffic control, and observability. We'll cover each area in depth in later sections.



The Istio architecture in 2020

The basis of an Istio service mesh is an Envoy proxy, which sits next to each service instance, and we call this a sidecar. This sidecar proxy is the foundation for the service mesh. It carries all the traffic in or out of your application, allowing it to generate metrics, apply policy, and control traffic flow. However, the more services that are deployed, the more sidecars in use, the complexity increases as do the number of problems that can occur. This is where Istio comes in.

Istio acts as the control plane, making each sidecar aware of the others and provides centralized control for all of them. We use the term "service mesh" because of Istio's power to program the sidecars, enforce policies, and collect telemetry.

Why a service mesh?

Microservices involve a lot of moving parts, and trying to control them at scale is like herding cats. Remember, the network is an inherent part of your application—an issue that didn't exist with monoliths. A service mesh is what's needed to consistently manage traffic, security, observability, and resilience. Now you're herding cats in a smaller pen where they're easier to get hold of. The service mesh moves these concerns into a common layer, out of the application and away from developers, so that a platform operator can manage them on behalf of the entire organization.

Service mesh key functions

To break it down to the key functions of the service mesh, it:

- **Allows developers to focus on the business:** There are so many things that can be removed from the developers' hands by moving to microservices and a service mesh. Because the sidecar proxy is handling so many things, including security, traffic and observability, you don't need to migrate existing libraries into new languages. It gives developers the opportunity to explore a variety of languages and frameworks to address software requirements.
- **Allows focus on the application capability:** Engineering time is better spent focusing on the application capability than, for example, determining the retry count for the calls and building

it into the application. The service mesh platform abstracts that from the developer. This being said, a service mesh will not hide bad code. In fact, with the mesh's observability features, bad code can be more obvious, so be warned!

- **Improves debugging:** The mesh handles each request individually, generating metrics and logging it individually, meaning there is great observability of the actions happening in your system. It helps you quickly identify whether an issue is application related or networking specific, and helps reduce time to resolution.
- **Improves operations**, allowing you to:
    - Use outlier detection (passive health checks) in Envoy to automatically remove unhealthy services from the load balancing pool to increase the overall availability and reliability of a service
    - Implement circuit breaking to avoid cascading failures caused by overloading service instances
    - Configure retry policies to resend failed requests to different service instances
    - Transparently observe your applications to get insight into unexpected dependencies and service communication failures
    - Reduce downtime with:
        - Canary release as a testing method for service updates, which allows you to roll out changes to a small part of your infrastructure and verify it works before pushing it to the rest of your production environment
        - Dictate retries, stating the number of times the request will try different endpoints, to make a successful connection
    - Use fault injection to test how your services would behave if there was a problem by deliberately forcing one into the system.
- **Improves security:**
    - The mesh ensures that the traffic moving between services is encrypted by default, with a strong (authenticatable) identity per application
    - Imposes policies at a granular level of service behaviors, not just connection level.
- **Enforces consistency:** App developers don't have to build any of the above into the logic anymore. The mesh does it for you. At a high level, the service mesh enables a services-first network. You get a uniform way to secure, manage, connect, and monitor your system, all done by the mesh.

The value of a service mesh stretches beyond just 'greenfield' microservice applications and is equally supportive of monoliths. Why? Monoliths aren't always a bad thing and suit certain business needs

well. In some cases, there's no need to consider a microservice application as a replacement. However, if your business is making the move to microservices, it'll take time and patience. More and more frequently, a service mesh is becoming a key consideration to help with the move by bridging legacy and new infrastructure.

How does a service mesh do it?

Let's look deeper into how it works.

Service mesh is a dedicated infrastructure layer that controls service-to-service communication over a network.

Secure Communication

A service mesh gives your organization powerful capabilities for enabling encryption in transit, enforcing service-to-service access policy, and auditing compliance to those policies.

mTLS

Istio provides every application a runtime identity in the form of a certificate. That certificate can be used at runtime to perform mutual Transport Layer Security (mTLS)—requiring both the client and the server to verify each other. This provides encryption in transit out of the box at greatly reduced operational burden, and serves as the basis for...

Authentication and Authorization of Service-to-Service Communication

Each service in the mesh is given a verifiable identity—the certificate we mentioned above—which is used at runtime to decide which of two services are allowed to communicate. Istio can go further, and apply policy per request on request metadata—like a JSON Web Token (JWT), the request's path, or other headers—in addition to coarse grained access based on identity.

Observability

If you're using any number of services, you need to know *exactly* what each of them are doing, and most importantly when they're not doing what they should. The very detailed observability that Istio enables lets you gain insight into your service's operations.

Metrics

Istio produces *operational metrics* about a service—the RED metrics: rate of **R**equests, rate of **E**rrors, **D**uration (as a distribution). These three metrics can be used to determine if most services are healthy or not without needing a deep understanding of the service's underlying business logic. Istio produces these metrics consistently, with identical dimensions, for every service in the mesh. This enables uniform tools (like dashboards) that work for all services.

Distributed Traces

Traces are a great way to understand end-to-end request flows through your system as a user experiences them. A service mesh can help with tracing by initiating traces at the edge of your infrastructure and propagating trace data into your applications. A mesh can't enable distributed tracing by itself, though: to really get distributed tracing you must update your applications to propagate trace data from incoming requests to outgoing requests itself. Most trace implementations have clients per language that will help you do this in your application.

Access Logs

Envoy can produce per-request access logs on behalf of your applications. These logs can include quite a lot of information about the request including the full Common Log format data, data about how Envoy itself processed the request, and more. This logging can be centrally configured and is consistent across the services in your mesh. This invariably allows you to build log processing pipelines that are universally usable by your application teams.

Traffic Management

A service mesh provides fine-grain control over traffic in your network. Today, you need to write code into your application to handle most of the considerations listed, but a service mesh moves this functionality out from a developer. Your organization can then centralize control in the hands of the platform team, or delegate control to your individual application teams (or, more likely, a mix of the two).

Live Configuration Update

In highly dynamic systems, it's critical that we be able to react to events quickly—at the speed of an API call, ideally. Istio and Envoy support live configuration reload without needing to restart. This

means the new configuration you push into your mesh goes into effect quickly. Compare that to today, where you have a bad retry loop in your code (because your retry logic probably is embedded in the code, probably as a for loop) requires a code change, rebuild, and redeploy of a binary. That's a far longer time to mitigate a problem than pushing updated configuration.

Fine-Grained Traffic Routing

Because a service mesh's sidecar intercepts your application's traffic, we can apply *client-side load balancing*. This means that individual instances of your service can make decisions like a traditional load balancer would, but per request rather than per connection. For example, based on an HTTP request's headers you can choose to send the request to a staging version of your service rather than the primary production version. Or you can split traffic by percentage to test a new version of your service incrementally ("canarying" your service). You can choose to route traffic based on just about any metadata about the request, including source or destination IP and port, HTTP headers, including cookies, JWTs, host headers, etc.

Resiliency

We've mentioned that a mesh provides many tools for resiliency. The client-side load balancing enabled by sidecar proxies let us perform retries with outlier detection (passive health checking) and circuit breaking, on each request your application sends. This makes your system far more reliable overall because individual instances of your service can react to transient failures independently and without you intervening. Together these features can greatly boost the success rate of calls to services (even external services you don't own).

Fault Injection

How does your UI behave when your access control service is under load and takes 5 seconds to return a result? What about when the backend it relies on fails at an elevated rate? Fault injection means deliberately introducing failures in an individual service so you can understand how the rest of your services behave in the presence of that failure. For example, you can configure a service to return a 500 response code, or add an additional 5 seconds of latency, to half of the requests it receives. This allows you to create consistent and repeatable tests that emulate past failures, or to reproduce live failures in a controlled way. This can be done with the push of a button, and doesn't require any changes to your application's code.

Traffic Shadowing

Traffic shadowing (*or request mirroring*) allows you to test new versions of a service with real production traffic rather than simulated or test traffic. Envoy does this by sending traffic to a secondary service in addition to its primary destination, but ignoring any results from the secondary service. This allows for more accurate and truer-to-life testing than many other methods. Be careful that your secondary service doesn't act on requests it receives in a way that interferes with the primary service!

Getting Started

We hope that this paper has given you an overview of what service mesh can do for you, but let's be clear that service mesh isn't going to solve all of your problems. An Istio and Envoy service mesh has a substantial collection of features and capabilities that should be used carefully and with consideration. Efforts to use all of them at once may not result in a successful implementation, so think about what you're trying to achieve, and what specific problems you're trying to solve. Start with something that you can test the value of, and move forward from there.

Tetrate has a wealth of resources, knowledge and expertise on Istio and Envoy. To find out more of what's available and what support we can offer, contact us at info@tetrate.io.

Learn more about Tetrate products: www.tetrate.io

Get started quickly with Tetrate

Enterprise ready service mesh for any workload on any environment

Contact Us | Schedule a Demo

About Tetrate

Tetrate enables a safe and fast modernization journey for enterprises. Built atop Envoy and Istio, its flagship product, Tetrate Service Bridge, spans traditional and modern workloads so customers can get consistent baked-in observability, runtime security, and traffic management—for all their workloads, on any environment. In addition to the technology, Tetrate brings a world-class team that leads the open Envoy and Istio projects, providing best practices and playbooks that enterprises can use to modernize their people and processes.

Location: Tetrate, 691 S Milpitas Blvd, Suite 217, Milpitas, CA 95035, USA

www.tetrate.io | info@tetrate.io

Copyright © 2020 Tetrate